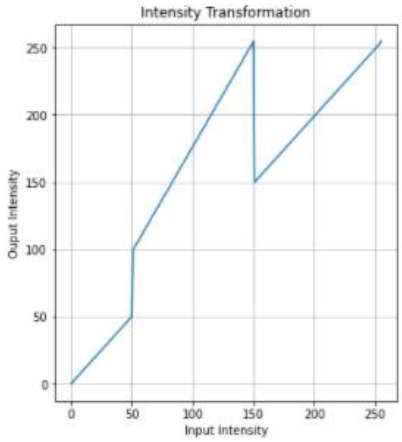
Assignment 01

D.M.S.S.DISSANAYAKE 190155L

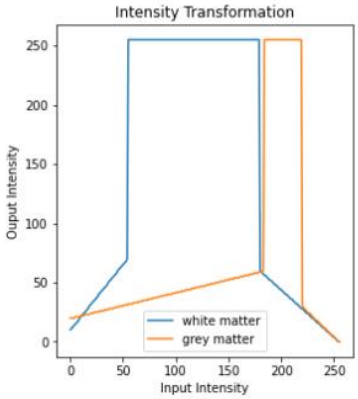
Q1)

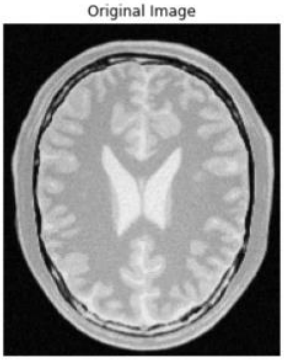



```
img = cv.imread("emma_gray.jpg", cv.IMREAD_GRAYSCALE)
assert img is not None

t1 = np.linspace(0,50,51)
t2 = np.linspace(100, 255, 100)
t3 = np.linspace(150,255,105)

t = np.concatenate((t1, t2, t3), axis=0).astype(np.uint8)

assert len(t) == 256
g = cv.LUT(img, t)

fig, ax = plt.subplots(1,3, figsize = (18,6))
ax[0].plot(t)
ax[0].set_title('Intensity Transformation')
ax[1].imshow(img, cmap= 'gray', vmin=0, vmax=255)
ax[1].set_title('Original Image')
ax[2].imshow(g, cmap= 'gray', vmin=0, vmax=255)
ax[2].set_title('Intensity Transformed Image')
plt.show()
```

As show in the Intensity Transformation, pixel with intensity range from 50 to 150 is mapped to a higher pixel intensity, while keeping the rest of the pixel intensities unchanged. As a result shown Intensity transformed Image each pixel in the *Original ima***ge** corresponding to the above range has a lighter in color.

Q2)





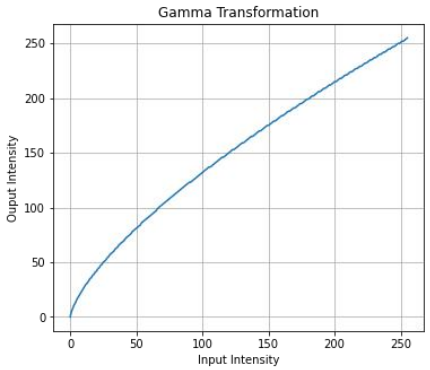

```
img = cv.imread("brain_proton_density_slice.png", cv.IMREAD_GRAYSCALE)
assert img is not None

t1 = np.linspace(10,70,55)
t2 = np.linspace(255, 255, 125)
t3 = np.linspace(60,0,76)
t4 = np.linspace(20,60,184)
t5 = np.linspace(255, 255, 36)
t6 = np.linspace(30,0,36)

t = np.concatenate((t1, t2, t3), axis=0).astype(np.uint8)
t_ = np.concatenate((t4, t5, t6), axis=0).astype(np.uint8)

assert len(t) == 256
g = cv.LUT(img, t)
g_ = cv.LUT(img, t_)

fig, ax = plt.subplots(1,4, figsize = (20,5))
ax[0].set_title('Intensity Transformation')
ax[1].imshow(img, cmap= 'gray', vmin=0, vmax=255)
ax[1].set_title('Original Image')
ax[2].imshow(g, cmap= 'gray', vmin=0, vmax=255)
ax[2].set_title('White Matter Accentuated image')
ax[3].imshow(g_, cmap= 'gray', vmin=0, vmax=255)
ax[3].set_title('Grey Matter Accentuated Image')
plt.show()
```

Accentuating White matter

Here the pixel intensities corresponding to white matter in the original image approximately falls in the range of 55 to 185. By Increasing the intensities of the pixels falling in the above mentioned range and giving a lowering intensity to pixels outside the range, We can accentuated the white matter by mapping the intensity transformation to the original image.

Like wise we can apply the suitable intensity transformation to accentuate the grey matter from the original Image.

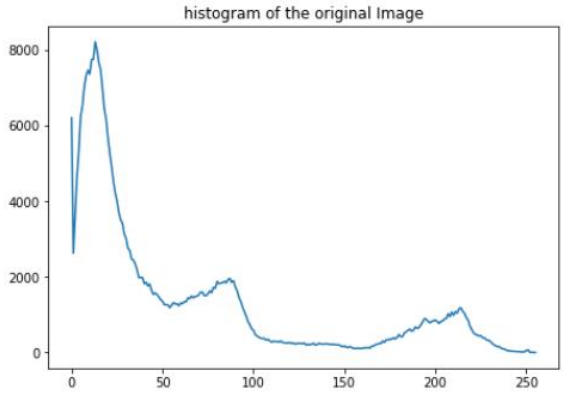
Q3)

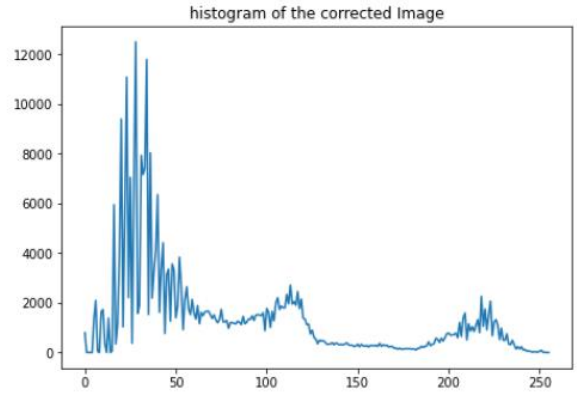



```
img = cv.imread("highlights_and_shadows.jpg", cv.IMREAD_COLOR)
assert img is not None

Lab = cv.cvtColor(img, cv.COLOR_BGR2Lab)
L,a,b = cv.split(Lab)
gamma = 0.7
t = np.array([(p/255)**gamma*255 for p in range(0,256)]).astype(np.uint8)
assert len(t) == 256
L_ = cv.LUT(L, t)

final_Lab = cv.merge((L_, a, b))
corrected_img = cv.cvtColor(final_Lab, cv.COLOR_Lab2BGR)

fig, ax = plt.subplots(1,3, figsize = (20,5))
ax[0].plot(t)
ax[0].grid()
ax[0].set_title('Gamma Transformation')
ax[1].imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
ax[1].set_title('Original Image')
ax[2].imshow(cv.cvtColor(corrected_img, cv.COLOR_BGR2RGB))
ax[2].set_title('Gamma Transformed Image')
plt.show()
```

- As shown in the gamma transformation, the intensity of the pixels with intensities closer to zero has been slightly increased. As a result (Shown in gamma transformed Image) the darker regions of the image will become slightly brighter.

- By applying this transformation, the details in the darker regions of the original image are more visible.



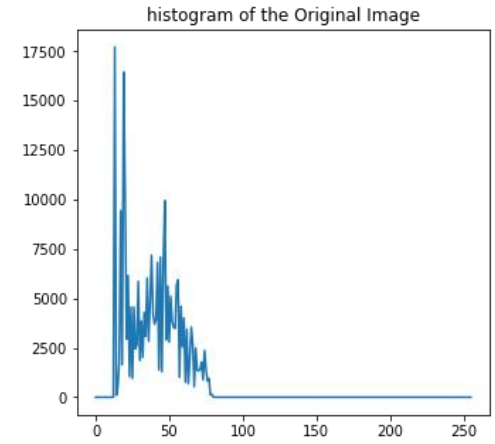


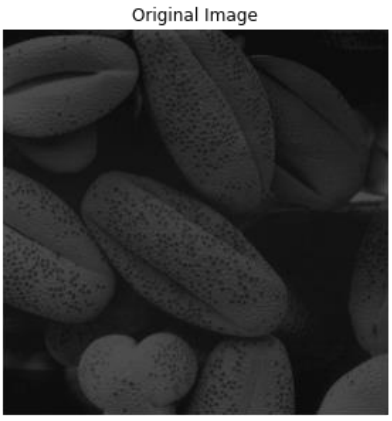

```
hist_img = cv.calcHist([img], [0], None, [256], [0,256])
hist_corrected_img = cv.calcHist([corrected_img], [0], None, [256], [0,256])

fig, ax = plt.subplots(1,2, figsize = (16,5))
ax[0].plot(hist_img)
ax[0].set_title('histogram of the original Image')
ax[1].plot(hist_corrected_img)
ax[1].set_title('histogram of the corrected Image')
plt.show()
```

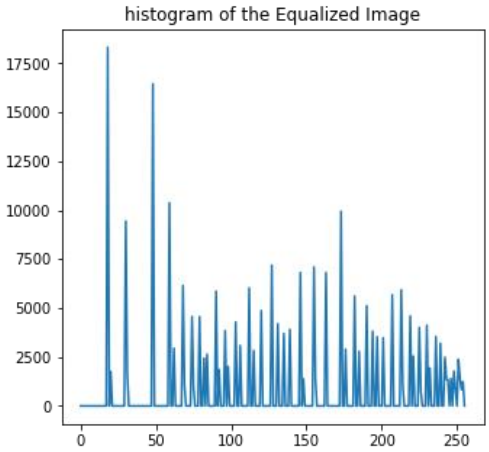
- By comparing the two histograms we can see that histogram of the corrected Image approximately corresponds to the right shifted histogram of the original image.

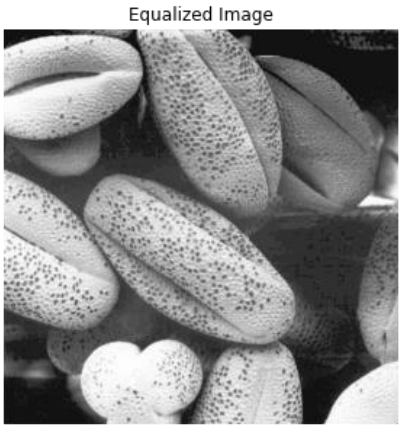
- The pixel concentration of lower intensities are moved to higher intensities.

Q4)










```
img = cv.imread('shells.png',cv.IMREAD_GRAYSCALE)
assert img is not None

s = np.zeros((256,), dtype=np.uint8)
sum = 0
for r in range(0,256):
    sum += np.count_nonzero(img == r)
    s[r] = np.uint8(sum*(255/img.size))

equalized_img = cv.LUT(img, s)
hist_img = cv.calcHist([img], [0], None, [256], [0,256])
hist_equalized_img = cv.calcHist([equalized_img], [0], None, [256], [0,256])
fig, ax = plt.subplots(2,2, figsize = (16,16))
ax[0,0].plot(hist_img)
ax[0,1].imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
ax[1,0].plot(hist_equalized_img)
ax[1,1].imshow(cv.cvtColor(equalized_img, cv.COLOR_BGR2RGB))
plt.show()
```

-In the histogram of the equalized image we can see that the no.of pixels are spreaded out over the range of 0 to 255 (full intensity range). Where as in the original image the concentration of pixels with lower intensities is greater (No pixels with intensities greater than 80). Hence it the original image is darker. The equalized image is brighter and the detail of the image is more visible.

Q5)

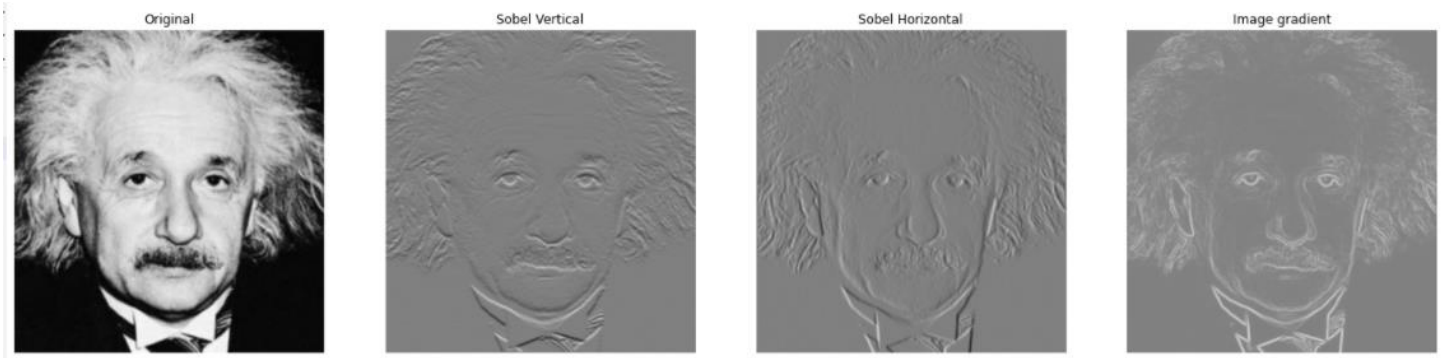
```
img = cv.imread('im01small.png', cv.IMREAD_GRAYSCALE)
img_large = cv.imread('im01.png', cv.IMREAD_GRAYSCALE)
scale = 4

def imageScale(img,scale,scaling_method):
    rows = int(scale*img.shape[0])
    columns = int(scale*img.shape[1])
    zoomed_img = np.zeros((rows,columns),dtype=img.dtype)

    if scaling_method == 'NN': # Nearest-neighbor
        for i in range(0,rows-1):
            for j in range(0,columns-1):
                x = i/scale
                y = j/scale
                img_x_pos = round(x) if(x<=img.shape[0]-1) else int(x)
                img_y_pos = round(y) if(y<=img.shape[1]-1) else int(y)
                zoomed_img[i,j] = img[round(img_x_pos),round(img_y_pos)]
    if scaling_method == 'BI': # bilinear Interpolation
        for i in range(0,rows-1):
            for j in range(0,columns-1):
                x,y = i/scale,j/scale
                x_left = math.floor(x)
                x_right = math.ceil(x) if(x<=img.shape[0]-1) else math.floor(x)
                y_top = math.floor(y)
                y_bottom = math.ceil(y) if(y<=img.shape[1]-1) else math.floor(y)
                r_y = (y-y_top)
                r_x = (x-x_left)
                zoomed_img[i,j] = math.floor((img[x_left,y_top]*(1-r_y) + img[x_left,y_bottom]*(r_y))*(1-r_x)
                + (img[x_right,y_top]*(1-r_y) + img[x_right,y_bottom]*(r_y))*(r_x))
    return zoomed_img

def SSD(original_img,zoomed_img):
    return np.square(original_img-zoomed_img).sum()/(original_img.shape[0]*original_img.shape[1])

zoomed_img_NN = imageScale(img,scale,"NN")
zoomed_img_BI = imageScale(img,scale,"BI")

print("Normalized SSD of nearest neighbour =", round(SSD(img_large,zoomed_img_NN),2))
print("Normalized SSD of bilinear interpolation =",round(SSD(img_large,zoomed_img_BI),2))

fig, ax = plt.subplots(1,2 , sharex = 'all' , sharey = 'all', figsize=(24,6))
ax[0].imshow(zoomed_img_NN,cmap= 'gray',vmin=0, vmax=255)
ax[0].set_title(' 4 x Zoomed image using Nearest neighbour')
ax[1].imshow(zoomed_img_BI,cmap= 'gray',vmin=0, vmax=255)
ax[1].set_title(' 4 x Zoomed image using bilineatr interpolation')
plt.show()
```

-The scaled Image of obtained from the nearest neighbor method is pixelated and can be seen clearly. Whereas in bilinear method the pixelation is not present but is has a slight blur in the scaled image.

-From the values of Normalized SSD we can say that bilinear Interpolation does a slightly better scaling compared to Nearest neighbor method.

```
Normalized SSD of nearest neighbour = 40.09
Normalized SSD of bilinear interpolation = 39.32
```

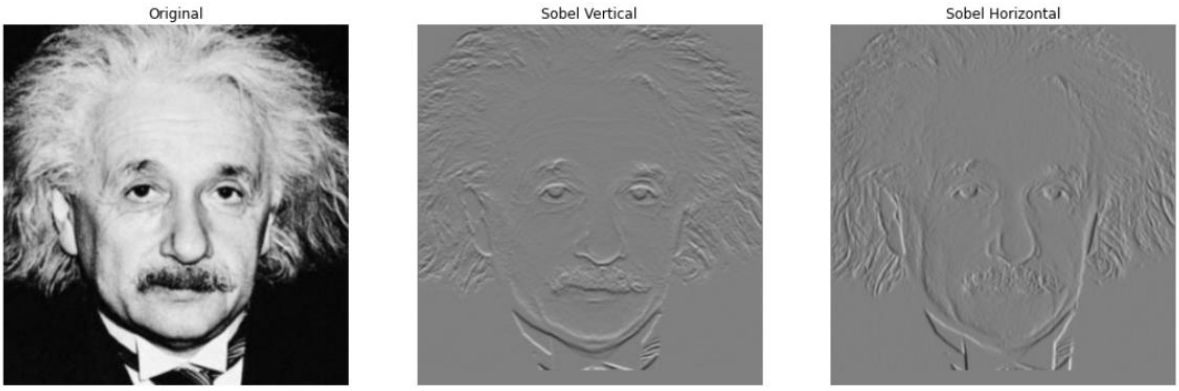
Q6) a




```
img = cv.imread(r'einstein.png', cv.IMREAD_GRAYSCALE).astype(np.float32)
assert img is not None

sobel_v = np.array([(-1,-2,-1),(0,0,0),(1,2,1)], dtype = np.float32)
img_x = cv.filter2D(img, -1, sobel_v)
sobel_h = np.array([(-1,0,1),(-2,0,2),(-1,0,1)], dtype = np.float32)
img_y = cv.filter2D(img, -1, sobel_h)
img_grad = np.sqrt(img_x**2+img_y**2)

sobel_filtered_img = np.zeros((img.shape[0],img.shape[1]), dtype=np.int16)
for i in range(1,img.shape[0-1]):
    for j in range(1,img.shape[1]-1):
        mat = img[i-1,j-1:j+2],img[i,j-1:j+2],img[i+1,j-1:j+2]
        sobel_filtered_img[i,j] = np.multiply(mat,sobel_v).sum()

fig, ax = plt.subplots(1,4 , sharex = 'all' , sharey = 'all', figsize=(24,6))
ax[0].imshow(img,cmap= 'gray',vmin=0, vmax=255)
ax[0].set_title('Original')
ax[1].imshow(img_x,cmap= 'gray',vmin=-1020, vmax=1020)
ax[1].set_title('Sobel Vertical')
ax[2].imshow(img_y,cmap= 'gray',vmin=-1020, vmax=1020)
ax[2].set_title('Sobel Horizontal')
ax[3].imshow(img_grad,cmap= 'gray',vmin=-1020, vmax=1020)
ax[3].set_title('Image gradient')
```

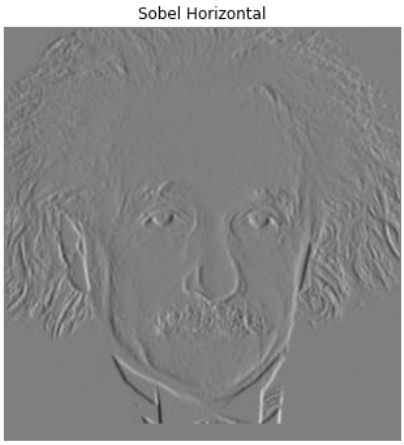
Q6) b

```
img = cv.imread(r'einstein.png', cv.IMREAD_GRAYSCALE).astype(np.float32)
assert img is not None

sobel_v = np.array([(-1,-2,-1),(0,0,0),(1,2,1)], dtype = np.float32)
sobel_h = np.array([(-1,0,1),(-2,0,2),(-1,0,1)], dtype = np.float32)

sobel_v_filtered_img = np.zeros((img.shape[0],img.shape[1]), dtype=np.int16)
sobel_h_filtered_img = np.zeros((img.shape[0],img.shape[1]), dtype=np.int16)

# Applying sobel operator using convolution
for i in range(1,img.shape[0-1]):
    for j in range(1,img.shape[1]-1):
        acc_sobel_v = 0
        acc_sobel_h = 0
        for m in range(sobel_v.shape[0]):
                for n in range(sobel_v.shape[1]):
                        acc_sobel_v += sobel_v[m,n]*img[i+1-m,j+1-n]
                        acc_sobel_h += sobel_h[m,n]*img[i+1-m,j+1-n]
        sobel_v_filtered_img[i,j] = acc_sobel_v
        sobel_h_filtered_img[i,j] = acc_sobel_h

fig, ax = plt.subplots(1,3 , sharex = 'all' , sharey = 'all', figsize=(18,6))
ax[0].imshow(img,cmap= 'gray',vmin=0, vmax=255)
ax[0].set_title('Original')
ax[1].imshow(sobel_v_filtered_img,cmap= 'gray',vmin=-1020, vmax=1020)
ax[1].set_title('Sobel Vertical')
ax[2].imshow( sobel_h_filtered_img,cmap= 'gray',vmin=-1020, vmax=1020)
ax[2].set_title('Sobel Horizontal')
```



Q6) c

```
img = cv.imread(r'einstein.png', cv.IMREAD_GRAYSCALE).astype(np.float32)
assert img is not None

sobel_h1 = np.array([[1], [2], [1]], dtype = np.float32)
sobel_h2 = np.array([1, 0,-1], dtype = np.float32)

sobel_h_filtered_img = np.zeros((img.shape[0],img.shape[1]), dtype=np.int16)

for i in range(1,img.shape[0-1]):
    for j in range(1,img.shape[1]-1):
        mat = np.array([img[i-1,j-1:j+2],img[i,j-1:j+2],img[i+1,j-1:j+2]])
        sobel_h_filtered_img[i,j] = np.array(sobel_h1*sobel_h2*mat).sum()

fig, ax = plt.subplots(1,2 , sharex = 'all' , sharey = 'all', figsize=(12,6))
ax[0].imshow(img,cmap= 'gray',vmin=0, vmax=255)
ax[0].set_title('Original')
ax[1].imshow(sobel_h_filtered_img,cmap= 'gray',vmin=-1020, vmax=1020)
ax[1].set_title('Sobel Horizontal')
```



Q7)

```
img = cv.imread(r'daisy.jpg', cv.IMREAD_COLOR)
assert img is not None

mask = np.zeros(img.shape[:2],np.uint8)
bgModel = np.zeros((1,65), np.float64)
fgModel = np.zeros((1,65), np.float64)

rect= (40,150,520,400)
cv.grabCut(img,mask,rect,bgModel,fgModel,5, cv.GC_INIT_WITH_RECT)
mask2 = np.where((mask==2) | (mask==0),0,1).astype('uint8')
mask3 = np.where((mask!=2) & (mask!=0),0,1).astype('uint8')
fg_img = img*mask2[:,:,np.newaxis]
bg_img = img*mask3[:,:,np.newaxis]

fig, ax = plt.subplots(1,3 , sharex = 'all' , sharey = 'all', figsize=(15,8))
ax[0].imshow(mask2)
ax[0].set_title('Final segmentation mask')
ax[1].imshow(cv.cvtColor(fg_img, cv.COLOR_RGB2BGR))
ax[1].set_title('foreground Image')
ax[2].imshow(cv.cvtColor(bg_img, cv.COLOR_RGB2BGR))
ax[2].set_title('Background Image')
```

Final segmentation mask

foreground Image

Background Image

```
img = cv.imread(r'daisy.jpg', cv.IMREAD_COLOR)
assert img is not None

mask = np.zeros(img.shape[:2],np.uint8)
bgModel = np.zeros((1,65), np.float64)
fgModel = np.zeros((1,65), np.float64)

rect= (40,150,520,400)
cv.grabCut(img,mask,rect,bgModel,fgModel,5, cv.GC_INIT_WITH_RECT)
mask2 = np.where((mask==2) | (mask==0),0,1).astype('uint8')
mask3 = np.where((mask!=2) & (mask!=0),0,1).astype('uint8')
fg_img = img*mask2[:,:,np.newaxis]
bg_img = img*mask3[:,:,np.newaxis]
sigma = 6
bg_img = cv.GaussianBlur(bg_img, (9,9),sigma)
enhanced_img = fg_img + bg_img

fig, ax = plt.subplots(1,2 , sharex = 'all' , sharey = 'all', figsize=(10,8))
ax[0].imshow(cv.cvtColor(img, cv.COLOR_RGB2BGR))
ax[0].set_title('Original Image')
ax[1].imshow(cv.cvtColor(enhanced_img, cv.COLOR_RGB2BGR))
ax[1].set_title('Enchanced Image')
```

Original Image

foreground Image

Why the edge is dark?

-When the gaussian blur is applied to the background image ,the intensity of the pixel locations corresponding to the edge of the foreground will be darker because in the background image the foreground region is not present and is darker. Therefore ,the edge pixels of the image when combined with the foreground image will be darker.

-